1 Coffee

A spectrograph shows the wavelengths absorbed by an object when exposed to energy. We can treat spectrographs as time series, thinking of wavelength as the time coordinate. The data sets `Coffee_train` and `Coffee_test` are on our course website, and contain spectrograph data from two varieties of coffee bean, Arabica and Robusta.

(a) Load the `Coffee_train` dataset. Use `pivot_longer` with the appropriate `names_prefix` to get the times out of the variable names and convert them to a numeric index variable. Convert the result to a tsibble with both ID and Class as key variables.

(b) Make a plot of all the series in this dataset. Use `facet_grid` to separate the Arabica and Robusta classes so you can compare them visually.

2 Feature extraction and PCA

(a) Compute the overall mean of each series. Make a dotplot colored to show the distribution of means for each Class of bean. Does the overall mean distinguish between the two classes?

(b) Compute the ACF features for all series and save this result.

(c) Use `prcomp` with `scale=TRUE` to perform principal components analysis (PCA) on the ACF features. Save this result: this is your PCA model.

(d) Inspect the coefficients of the 1st principal component (PC1).

(e) Use `broom::augment` to compute the PC decomposition for all training series. It takes the PCA model and the ACF features data as its two arguments.

(f) Make a dotplot colored to show the distribution of PC1 for each Class of bean. Does PC1 distinguish between the two classes? Where is the cut value on the x-axis that separates the two classes?

3 Classification using the first principal component

Now we'll use the PC decomposition of ACF features to classify unknown coffee spectrographs as Arabica or Robusta. We used the train data to build the model, and we use the test data to see how the model performs.

(a) Load the `Coffee_test` dataset, and clean it up the same way you did with the training data.

(b) Compute the ACF features for all test series and save this result.

(c) Using the PCA model from the **training** data, use `broom::augment` with the `newdata` argument equal to your **test** data. This calculates the principal components of the test data.

(d) Classify all of the test series by whether their value of PC1 is above or below the cut value from 2(f). For the test data, how many points does the model classify correctly vs incorrectly?

4 FordA

The data sets `FordA_train` and `FordA_test` are on our course website. Each series is an engine noise recording, and there are two classes: properly working engines and engines with a problem. This is a much larger and more difficult classification problem than Coffee.

(a) Load and clean the training data. Your code from the Coffee data should work almost without change. Since the classes in FordA are 1 and -1, you may find it helpful to convert the Class variable to a factor type.

(b) Make a plot so you can visually compare the two classes of engine noise.

5 Classification using KNN in feature space

k-nearest-neighbors (KNN) classification chooses a class for a point based on the classes of its k nearest neighbors in the training data. In this model, we'll classify points by their one nearest neighbor in the 6-dimensional space consisting of the six ACF features.

(a) Compute the ACF features for the training data. Make scatterplots of some pairs of these features, colored by engine class. You should see that the features do a reasonable job of separating the two classes.

(b) Load the FordA test data and compute its ACF features.

(c) The `class::knn` function computes the KNN classification for points in the test set, given points in the train set. `class::knn` expects your data to be a matrix of purely numeric values. You'll need to extract the six numeric ACF feature columns from your train data and from your test data, and pass those as the `train` and `test` arguments to `knn`. The `cl` argument should get the Class variable from your training data. The `knn` function returns the classifications for the test set.

(d) Check the KNN classification for your test data against the actual Class variable of your test data. What percentage did this process classify correctly?

(e) Does adjusting the value of k in the KNN classification make a difference?